Lecture 6 Object Orientation

Class Implementation
Concepts:
  Encapsulation
  Inheritance
  Polymorphism

Template/Syntax:

```csharp
Public Class ClassName (naming: CPoint)
End Class
```

Elements of the class:

- **Data**
  1. Internal data
     - private; you cannot access internal data from outside the class
     - naming convention: m____
  2. Properties
     - way to retrieve (get) and change (set) value of properties

- **Behaviors**
  3. Methods
  4. Events

*Use the keyword “property” to declare a property of a class.*

1. **Properties**

   - Internal data: m____

   ```csharp
   Private mX, mY As Integer
   Public Property X() As Integer
     Get
       Return mX
     End Get
     Set (ByVal Value As Integer)
       mX = Value
     End Set
   End Property
   
   Notes: mX is the internal variable that stores private information.
   ‘Set’ is used to assign a value to a property of a class.
   ‘Get’ is used to retrieve the value of the property.*
Information hiding, keep internal data private. Use external methods to access the data, either to view it or to change it.

Instantiate an object of this class type. The object with have properties and methods associated with it, because it is an instance of the class.

Example: If you were to create a Textbox Class, you would need to declare a size property. You would use the ‘get’ method to retrieve the size. You would use the ‘set’ property to change the default size.

Instantiate the class:
   Dim objF1 as new CPoint

Access the property of the class:
   objF1.X

You can create a property that is read or write only by adding the keyword “ReadOnly” or “WriteOnly” to the declaration statement. Example:

   Public ReadOnly Property X() As Integer

If you do not declare ReadOnly or WriteOnly, then it is both readable and writeable.

2. Methods

   Public Function distance() As Single
       Return Math.Sqrt(mX^2 + mY^2)
   End Function

Overloading:
You can use the same name for a method, but have different parameters, and the methods can perform totally different tasks.

Example:

Public Overloads Function validate(ByVal a As String)
Public Overloads Function validate(ByVal a As Integer)

validate(x)
The code is smart enough to use the first function, if x is a string and the second function if x is an integer.

To overload, you must use the keyword “Overloads” in your function declaration statement.

Private Overloads Function CheckData(ByVal ssn As String) As _
Boolean

Private Overloads Function CheckData(ByVal age As Integer) As Boolean

You can overload the constructor without using the keyword “overloads”

Constructor:

Public Sub New()
    mX = 0 'assign default values for internal data
End Sub

Public Sub New(ByVal xpos As Integer, ByVal ypos As Integer)
    mX = xpos
End Sub

Dim aPt As New CPoint(3, 4)

To create a new class:
1. Project ➔ Add Class
2. Change class name
3. Create internal (private) variables to store data (follow the naming convention, m___)
4. Declare (public) properties of your class. All properties that you declare need a get and a set method, so that people outside the class can see the property and change the property, unless they are ReadOnly or WriteOnly properties. People will see the X property by using the get method. They will change the X property by using the set method.

Public Property X() as Integer
    Get
        Return mX
    End Get
    Set(ByVal Value As Integer)
        mX = Value
    End Set
End Property

5. Declare (public) functions. Example:

Public Function distance() as Single
    Return Math.Sqrt(mX^2 + mY^2)
End Function
6. Create constructor method for your class.
   Public Sub New()
       mX = 0
       mY = 0
   End Sub
7. Create override constructors, if desired:
   Public Sub New(ByVal xValue As Integer, ByVal yValue As Integer)
       mX = xValue
       mY = yValue
   End sub
8. Optionally, create destructor method for your class (use the Finalize method)

Now, you can create an instance of the class that you just defined. There are several ways to do this:

1. Dim aPt as New CPoint  ‘This uses the default constructor.
2. Dim bPt as New CPoint(3, 4)  ‘This uses the override constructor
3. Dim cPt as CPoint
   cPt = new CPoint(5,12)

Now, you can access the X property of your aPt object:
   MsgBox(aPt.X)

Now, you can change the value of the X property and see that it has changed:
   aPt.X = 5
   MsgBox(aPt.X)

You can also use the method that we created:
   MsgBox(cPt.distance())

Garbage Collection:
If you have an object that has been created that you no longer need, then it should be destroyed so that the memory being used by the object is released.

VB has automatic garbage collection. In C++, you have to destroy all of the objects yourself.

If you want to destroy an object, then you use the destructor. To do this, you assign nothing to your object and the garbage collector will know that it is garbage. Ex:  cPt = Nothing

You can also ask the garbage collector to collect the garbage now, by using: System.GC.Collect()
When the garbage collector collects, it executes the Finalize() method.
Ex: We have a CPoint class, and we have three instances of the CPoint class called aPt, bPt, and cPt. Suppose we are done with cPt and no longer need it. Then we can destroy it by assigning Nothing to cPt:   cPt = Nothing.

Then, the garbage collector comes, destroys the cPt object, and calls the Finalize() method. Because we now have one fewer object, we want to update the count of the number of objects:

```
Garbage Collection --> Object  \rightarrow  Finalize (Destructor)
```

```
Public Overrides Sub Finalize()
    mCount -= 1  'Comment: mCount = mCount - 1
End Sub
```

See demo LD07-1

1. Create the CPoint class
   Project \rightarrow Add Class

2. Create the public properties of the class and their methods of access (‘Get’ and ‘Set’ methods)
   Public Property X() as Integer
      Get
      Return mX
      End Get
      Set (ByVal Value As Integer)
      mX = Value
      End Set

   Public Property Y() as Integer
      Get
      Return mY
      End Get
      Set (ByVal Value As Integer)
      mY = Value
      End Set

3. Create a public function called distance()
   Public Function distance() as Single
      Return Math.Sqrt(mX^2 + mY^2)
   End Function

4. Create additional constructors
   Public sub new(byval xValue as integer, byval yValue as integer)
mX = xValue
mY = yValue
End sub

5. Create the shared property called Count()
   Public Shared ReadOnly Property Count() As Integer
       Get
       Return mCount
       End Get

6. Go back and add the count functionality to each of your constructors, because every
time a new instance of this class gets created, we want to add one to our counter. Add
this line of code to each constructor:
   mCount = mCount + 1    ‘Or:   mCount += 1

7. Update the Finalize() method so that when an object gets destroyed by the garbage
   collector, that you reduce the count of active objects. Add the following line of code to
the Finalize() method:
   mCount = mCount – 1  ‘Or:  mCount -= 1
   MsgBox(mCount)   ‘To display the count of objects

See code in Demo LD07-1.

The class that we created can be used within this application only, unless we put our class
into the class library.

To make this new class accessible by other projects, we put it into the class library.

The NameSpace is used to organize the classes within the library. You store your class in
a name space.

CLASS LIBRARY
   Class Library ➔ Reusable Code
      (Use Build Solution) ➔ Compiled Classes ➔ Assembly (.dll)

   NAMESPACES

      Namespace newNameSpace

      Class

      End Class

      End Namespace
Add Reference ➔ Imports

Recommended Naming Convention:

Company\text{\textunderscore}Technology\text{\textunderscore}Name.

\text{\textit{root}}

1. New Project ➔ Class Library (instead of Windows Application)
2. Name your new class library. Ex: CShapeLib
3. Above the class declaration, type your: “Namespace CShapeLib” where CShapeLib is the name of your technology.
4. Below the “End Class,” type: “End Namespace”
5. Project ➔ (Name of your project) Properties ➔ Root Namespace ➔ \textit{enter your company name}
6. Build ➔ Build Solution, to build the Assembly code that is needed. It will be a .ddl file. Assembly code is an executable file.

Call our new class (that is stored in the class library) from within another application:

1. First, you need to create a reference from your new project to your technology class that is stored in the class library. To do this, go to Project ➔ Add Reference ➔ Browse ➔ CShapeLib (or your technology name) ➔ CShapeLib.ddl

2. You then need to import the class into your project. Above the class declaration, type:

\text{imports UWMSIS.CShapeLib}

3. Now, within your new project, you can declare a new object using the CPoint class:

\text{Dim aPt As New CPoint(3, 4)}

\textbf{INHERITANCE}

With inheritance, you are inheriting common data, properties, methods, functions, and events. Then, within your new class, you add the properties, methods, functions, data that are unique to this new class. For example, if you want to create a circle class, you can inherit the the center point of the circle from the CPoint class, and then add a radius property to your circle class.

Suppose I want many different classes:

\begin{verbatim}
CPoint

| CCircle         CSquare
| CCylinder       CDoughnut
\end{verbatim}
A circle differs from a point in that a circle has a radius. A circle also has an area.

We will demonstrate inheritance by looking at the CPoint, CCircle, and CCylinder classes.

We will demonstrate polymorphism by looking at the CSquare and CDonut classes.

We will also talk about the difference between Overriding and Overloading.

At the end of the CPoint class definition, declare another class called CCircle that has CPoint as its parent class:

```
Public Namespace CShapeLib
    Public Class CCircle
        Inherits CPoint

        Private mRadius As Single

        Public Property Radius() as Single
            Get
                Return mRadius()
            End Get
            Set(ByVal Value as Single)
                mRadius = Value
            End Set
        End Property

        Public Function area() as Double
            Return Math.PI * mRadius^2
        End Function

    End Class
End Namespace
```

Example:
Base Class \( \rightarrow \) CAccount

Derived Class \( \rightarrow \) CChecking, CSavings

Class CSavings
    Inherits CAccount

There are four keywords:
Private: only visible within the same class
Public: available everywhere
Protected:  visible within the same class and within derived classes (meaning classes that inherit this class) Watch out: You can easily lose control of the data in your parent class! Try to avoid changing private to protected! (To do this, call the constructor of the parent class by using the MyBase keyword. See example below.)

Friend: visible within the project (and assembly)

Use the MyBase keyword to refer to something in the parent class. For example, within the constructor of your circle class, you can call the parent class’ constructor by writing: MyBase.New(xValue, yValue)

You can then assign a default value to the radius of your new circle:

mRadius = 0

Don’t forget to compile your code by going to: Build ➔ Build Solution

Overriding:
Suppose we have an area() function in CPoint class that returns 0. Suppose the area function is overridable. To make it overridable, you declare the function like this (within the parent class):

Public Overridable Function area() As Double
    Return 0
End Function

Suppose the area is calculated differently within the circle CCircle class.

Declare the area function within the CCircle class using the keyword overrides:

Public Overrides Function area()
    Return Math.PI * mRadius ^ 2
End Function

POLYMORPHISM
One method call can result in different actions occurring, depending on the type of object.

A method of the same name acts differently depending on what type of object it is acting upon. For example, the area function performs a different calculation on a square object, than it does on a circle object.

Suppose our CPoint parent class has two children classes: CCircle and CSquare.

A circle “is a” point and a square “is a” point, but a point is not necessarily a circle, nor is a point necessarily a square.

Dim aList As CPoint
Dim aCir As New CCircle(2,3,1)
aList = aCir    ‘You are allowed to assign a child object to a parent object.
Suppose we have a list of all of the objects that have been declared. Suppose all of the objects are stored in an array called aList. The pseudocode for this is:

\[
\begin{align*}
    \text{aList(0)} &= \text{new CPoint(2,3)} \\
    \text{aList(1)} &= \text{new CCircle(2,3,1)} \\
    \text{aList(2)} &= \text{new CSquare(3,4,1)}
\end{align*}
\]

Then, you can loop through all objects in your area and calculate their area, even if the area calculation differs, depending on the object. For example,

\[
\begin{align*}
    \text{For index = 0 To 2} \\
    &\quad \text{MsgBox(aList(index), area)} \\
    \text{Next index}
\end{align*}
\]

Other examples of polymorphism:

- **Parent Class:** Employee
- **Child Classes:** Hourly Employee, Salaries Employee, Manager Employee

Each of these child classes inherits the basic attributes of the employee class, but then you add additional attributes to them that are specific to those types of employees.